

**Course Code : CBS3007**

**Course: Data Mining and Analytics**

Alan Thomas

21BBS0115

Github link for the datasets and code-

<https://github.com/ALANT535/DATA-MINING-RESOURCES/tree/main/DA2>

# Aim

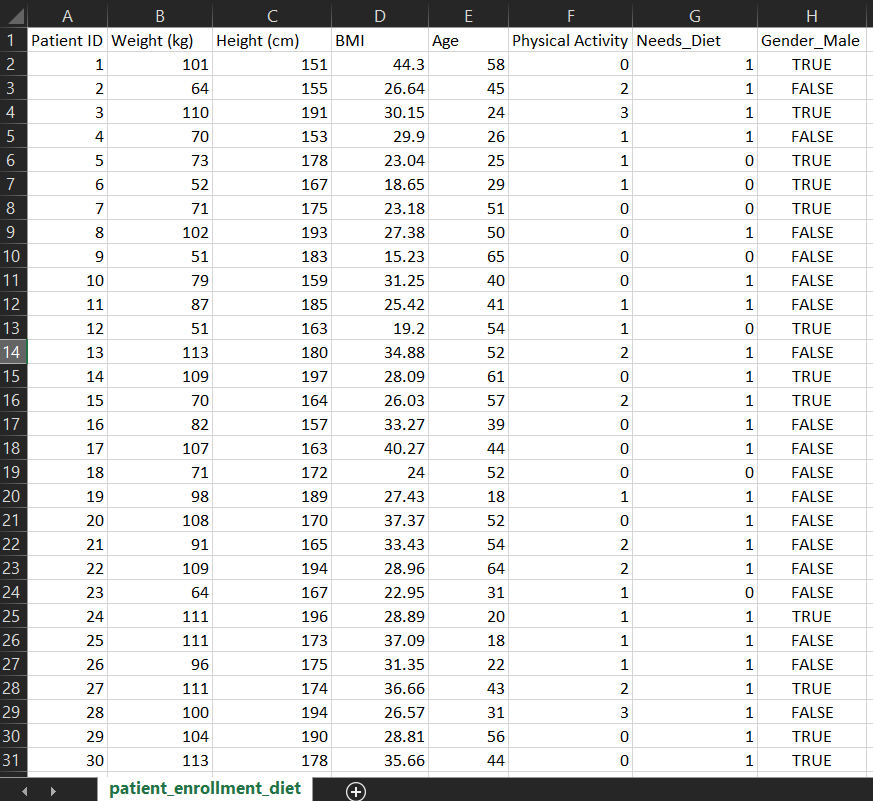
To Collect the data set consists of 50 observations about patient enrolment in diet maintenance based on gender, weight, BMI etc (minimum 7 features). Implement a model that will recommend a strict diet is necessary or not for a patient using the naïve Bayes classification algorithm.

LIBRARIES USED: Pandas, Numpy, Scikit Learn

Dataset : https://github.com/ALANT535/DATA-MINING-RESOURCES/tree/main/DA2/Q1

# SECTION 1

Sample Input



Code

import pandas as pd

from sklearn import naive\_bayes

from sklearn import model\_selection

from sklearn import metrics

from sklearn import preprocessing as pp

import numpy as np

data = pd.read\_csv('patient\_enrollment\_diet.csv')

# encoding

le = pp.LabelEncoder()

data['Physical Activity'] = le.fit\_transform(data['Physical Activity'])

temp = list(data.columns)

temp[7] = 'Physical Activity'

data.columns = temp

data = pd.get\_dummies(data, columns=['Gender'], drop\_first=True)

y = data['Needs\_Diet']

X = data.drop(['Needs\_Diet' , 'Patient ID'],axis = 1)

X\_train, X\_test, y\_train, y\_test = model\_selection.train\_test\_split(X, y, test\_size=0.25, random\_state=42)

nb = naive\_bayes.GaussianNB()

nb.fit(X\_train,y\_train)

y\_pred = nb.predict(X\_test)

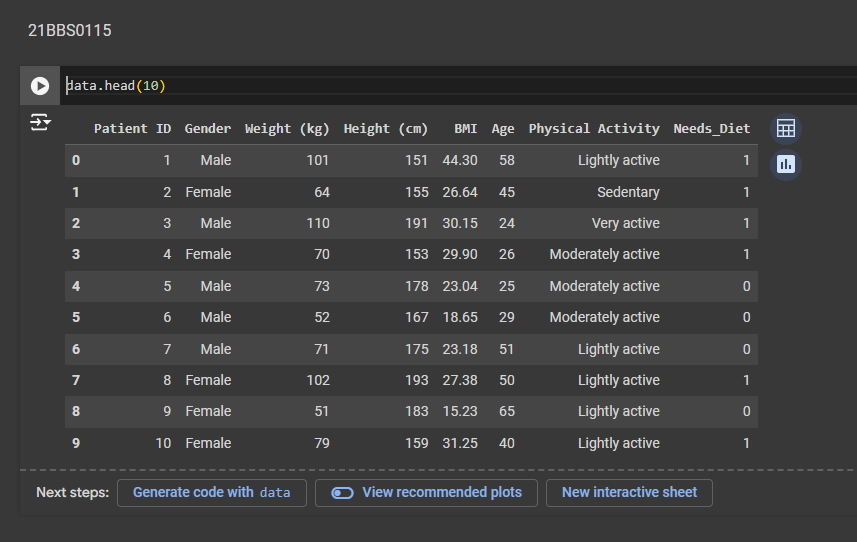
print('accuracy score:' , metrics.accuracy\_score(y\_test, y\_pred))

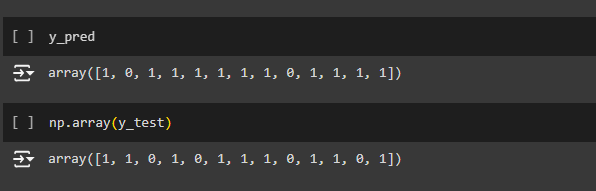
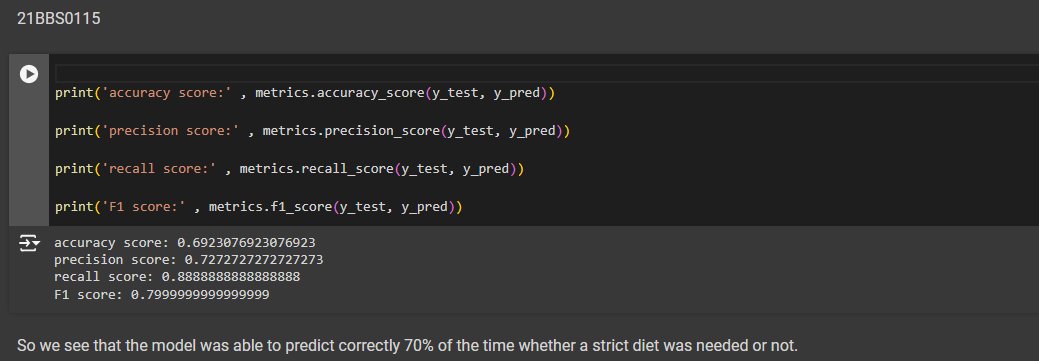
print('precision score:' , metrics.precision\_score(y\_test, y\_pred))

print('recall score:' , metrics.recall\_score(y\_test, y\_pred))

print('F1 score:' , metrics.f1\_score(y\_test, y\_pred))

Output





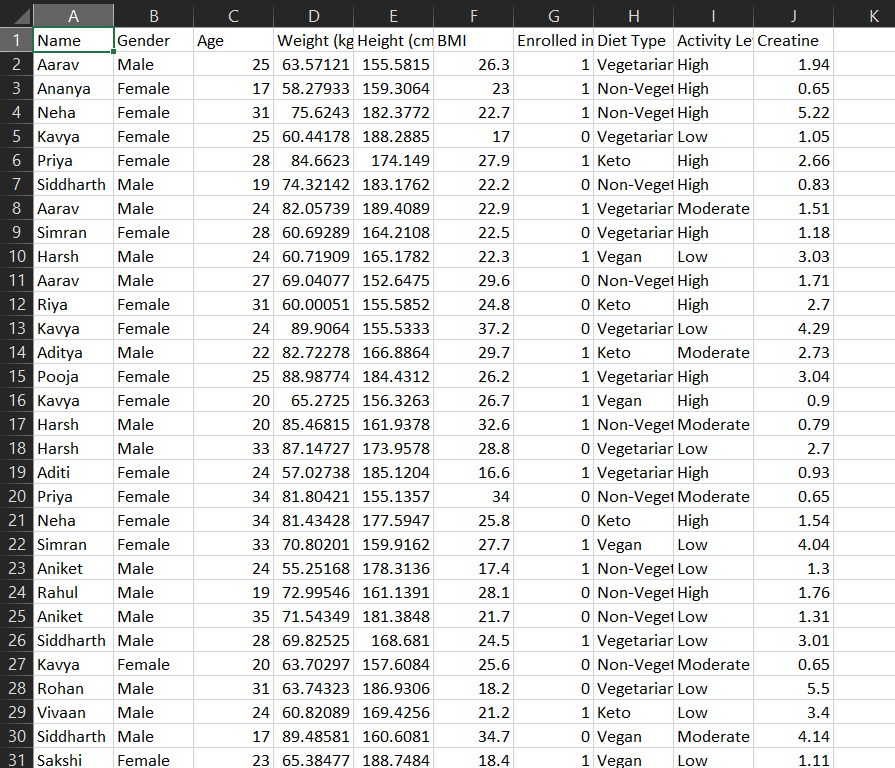
**SECTION 2**

Aim : To Implement K-means method of clustering and use the patient details data set to classify into 3 clusters such as a person is normal, healthy and weak. A person must be clustered as any one of normal/healthy or weak based on their input values.

Libraries : Numpy, Pandas, sklearn, seaborn

Dataset : https://github.com/ALANT535/DATA-MINING-RESOURCES/tree/main/DA2/Q2

Sample Input



# Code

import pandas as pd

from sklearn import naive\_bayes

from sklearn import model\_selection

from sklearn import metrics

from sklearn import preprocessing as pp

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

from sklearn.preprocessing import StandardScaler

import numpy as np

import seaborn as sns

data = pd.read\_csv('patient\_dataset.csv')

data.head(10)

data = data.drop(['Name' , 'Gender' , 'Age' , 'Enrolled in Diet Plan' , 'Diet Type' , 'Activity Level'] , axis = 1)

# check missing vlaues

data.any().isna().sum()

data.head()

num\_clusters = [i for i in range(2, 7)]

def kmeans\_inertia(num\_clusters, x\_vals):

inertia = []

for num in num\_clusters:

kms = KMeans(n\_clusters=num, random\_state=42)

kms.fit(x\_vals)

inertia.append(kms.inertia\_)

return inertia

X\_scaled = StandardScaler().fit\_transform(data)

inertia = kmeans\_inertia(num\_clusters,X\_scaled)

inertia

kmeans3 = KMeans(n\_clusters=3, random\_state=42)

kmeans3.fit(X\_scaled)

data['cluster'] = kmeans3.labels\_

def give\_label(cluster\_num):

if cluster\_num == 0:

return 'Weak'

elif cluster\_num == 1:

return 'Normal'

else:

return 'Healthy'

data['Class'] = data['cluster'].apply(give\_label)

import matplotlib.pyplot as plt

df = data.copy()

features = ['BMI', 'Creatine']

X = df[features]

# Plotting the clusters

plt.figure(figsize=(10, 8))

for cluster in df['cluster'].unique():

cluster\_data = df[df['cluster'] == cluster]

plt.scatter(cluster\_data['BMI'], cluster\_data['Creatine'], label=f'Cluster {cluster}', s=50, alpha=0.6)

plt.xlabel('BMI')

plt.ylabel('Creatine')

plt.title('Clusters Visualization with BMI and Creatine')

plt.legend()

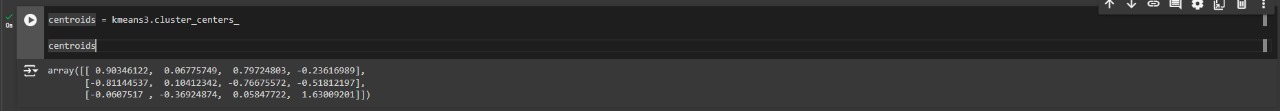
plt.grid(True)

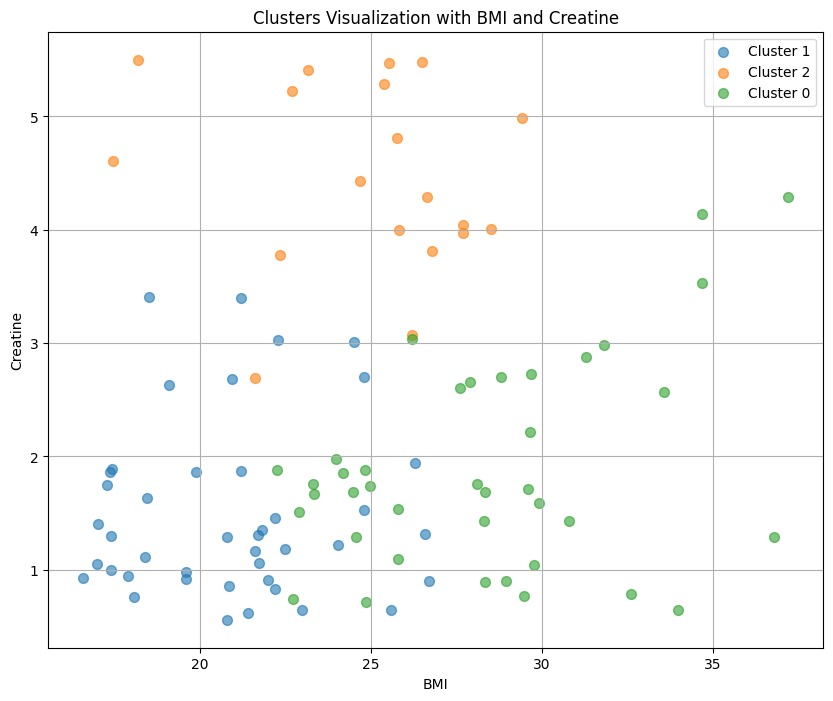
plt.show()

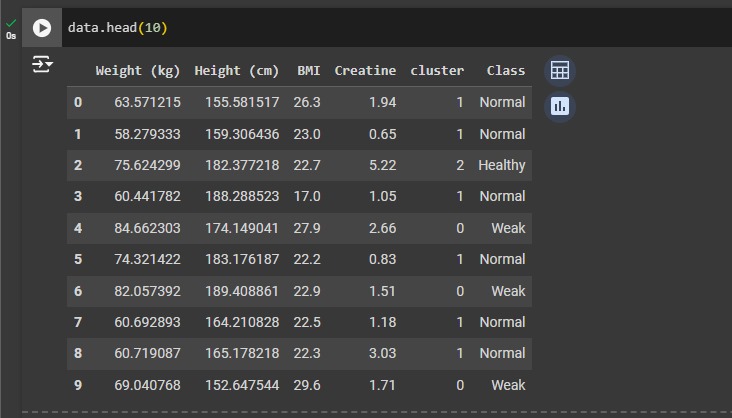
centroids = kmeans3.cluster\_centers\_

print(centroids)

Output :







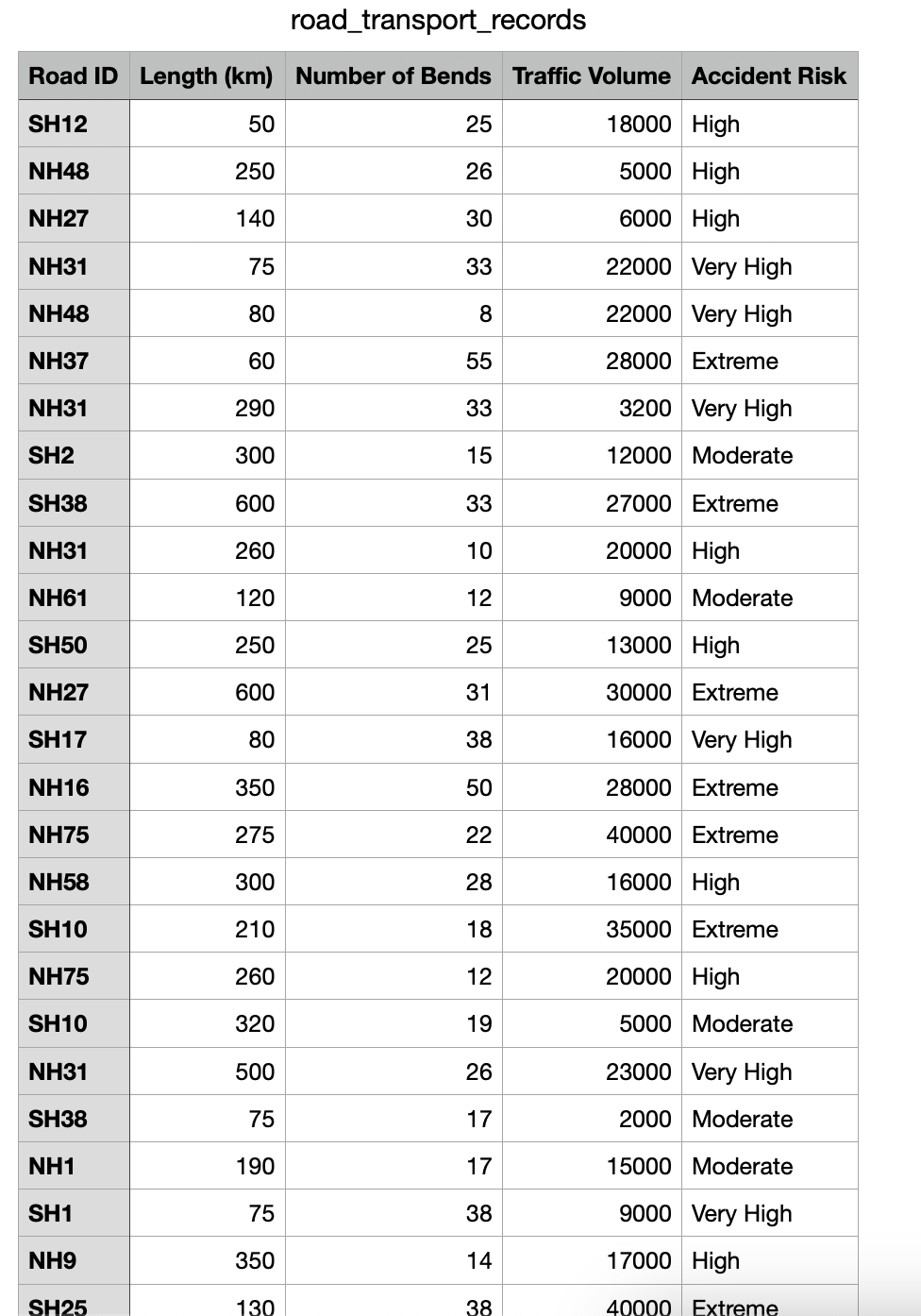
# SECTION 3

Aim: Implement the ID3 algorithm on the dataset to recommend the decision tree to classify the data.

Libraries Used : Numpy, Pandas, sklearn, matplotlib, seaborn

Dataset : https://github.com/ALANT535/DATA-MINING-RESOURCES/tree/main/DA2/Q3

Sample Input



**Code and Output Code**

Code :

import pandas as pd

import math

import matplotlib.pyplot as plt

import networkx as nx

from networkx.drawing.nx\_agraph import graphviz\_layout

df = pd.read\_csv(r"DA2\Q3\road\_transport\_records.csv")

def calculate\_entropy(data, target\_column):

total\_rows = len(data)

target\_values = data[target\_column].unique()

entropy = 0

for value in target\_values:

value\_count = len(data[data[target\_column] == value])

proportion = value\_count / total\_rows

entropy -= proportion \* math.log2(proportion) if proportion != 0 else 0

return entropy

def calculate\_information\_gain(data, feature, target\_column, entropy\_outcome):

unique\_values = data[feature].unique()

weighted\_entropy = 0

for value in unique\_values:

subset = data[data[feature] == value]

proportion = len(subset) / len(data)

weighted\_entropy += proportion \* calculate\_entropy(subset, target\_column)

information\_gain = entropy\_outcome - weighted\_entropy

return information\_gain

def id3(data, target\_column, features):

if len(data[target\_column].unique()) == 1:

return data[target\_column].iloc[0]

if len(features) == 0:

return data[target\_column].mode().iloc[0]

entropy\_outcome = calculate\_entropy(data, target\_column)

best\_feature = max(features, key=lambda x: calculate\_information\_gain(data, x, target\_column, entropy\_outcome))

tree = {best\_feature: {}}

features = [f for f in features if f != best\_feature]

for value in data[best\_feature].unique():

subset = data[data[best\_feature] == value]

subtree = id3(subset, target\_column, features)

tree[best\_feature][value] = subtree

return tree

def plot\_tree(tree, parent\_name, graph, depth, max\_depth):

if depth > max\_depth:

return

if isinstance(tree, dict):

feature = list(tree.keys())[0]

for value, subtree in tree[feature].items():

node\_name = f"{feature} = {value}"

graph.add\_node(node\_name)

graph.add\_edge(parent\_name, node\_name)

plot\_tree(subtree, node\_name, graph, depth + 1, max\_depth)

else:

leaf\_name = f"Accident Risk: {tree}"

graph.add\_node(leaf\_name)

graph.add\_edge(parent\_name, leaf\_name)

def visualize\_decision\_tree(decision\_tree, max\_depth=3):

graph = nx.DiGraph()

root\_name = list(decision\_tree.keys())[0]

graph.add\_node(root\_name)

plot\_tree(decision\_tree, root\_name, graph, 0, max\_depth)

plt.figure(figsize=(20, 15))

pos = nx.spring\_layout(graph, seed=42, k=0.5, iterations=50)

nx.draw(graph, pos, with\_labels=True, node\_size=3500, node\_color="lightblue", font\_size=12, font\_weight="bold", arrows=True, connectionstyle='arc3,rad=0.1')

plt.title("Decision Tree Visualization")

plt.show()

features = ['Length (km)', 'Number of Bends', 'Traffic Volume']

decision\_tree = id3(df, 'Accident Risk', features)

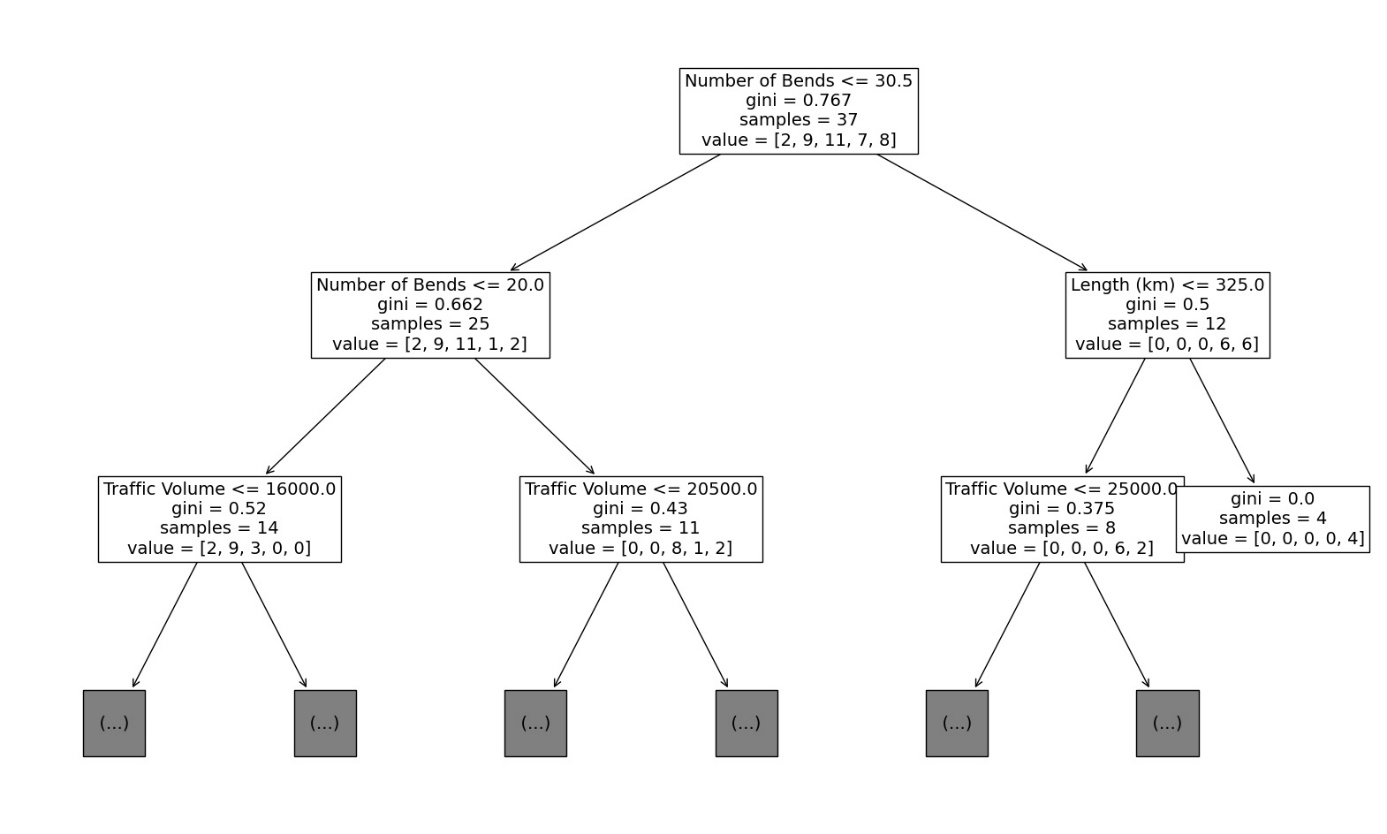
print("\nGenerated Decision Tree using ID3 algorithm:")

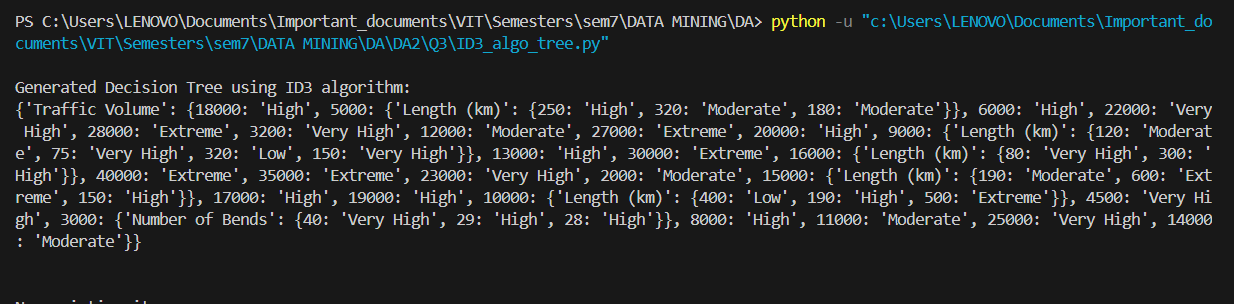
print(decision\_tree)

print("\n\nNow printing it-")

visualize\_decision\_tree(decision\_tree)

Output :





RESULT:

Created the Decision Tree based on the concept of ID3 algorithm.

**XXXXXXXXXXXXXXX**